ปราสาททางเดินลับ (Castle)

1 second, 512 MB

ปราสาทแห่งหนึ่งมีห้องจำนวน N ห้อง เนื่องจากเจ้าของปราสาทชอบเรียนวิทยาการคำนวณมาก จำนวนห้อง N จึงเท่ากับ $2^b - 1$ สำหรับบางจำนวนเต็ม b > 0

มีทางเดินเชื่อมระหว่างห้องเหล่านี้จำนวน M เส้น โดยที่ N – 1 <= M <= N + 9 เส้น เพื่อความสมมาตรจะมีทางเดินแบบธรรมดาจำนวน N – 1 เส้น เชื่อมห้องเข้าด้วยกันเป็นโครงสร้างแบบต้นไม้สมบูรณ์ (full binary tree) กล่าวคือมีห้องหนึ่งที่เป็นเสมือนราก (root) ของต้นไม้นี้ จากนั้นห้องจะแบ่งเป็นชั้น ชั้นที่ 1 มี 2 ห้อง ชั้นที่ 2 มี 4 ห้อง และจะมีจำนวนห้องเพิ่มในแต่ละชั้นขึ้นสองเท่าเรื่อย ๆ จนครบ N ห้อง แต่ละห้องที่ชั้นที่ a จะมีทางเดินเชื่อมไปอีกสองห้องที่ชั้นที่ a+1 สำหรับทางเดินพิเศษอีก M – N + 1 <= 10 เส้นเป็นจะเป็นทางลัดเชื่อมระหว่างคู่ของห้องอื่น ๆ

ในบางกรณีทดสอบคุณจะทราบว่าทางเดินชุดใดเป็นทางเดินธรรมดา บางกรณีทดสอบคุณจะไม่ทราบว่าทางเดินใดเป็นทางเดินธรรมดาบ้าง

ทางเดินต่าง ๆ ในปราสาทก็ผุพังไปตามกาลเวลา ให้คุณเขียนโปรแกรมรับข้อมูลการพังของทางเดินต่าง ๆ พร้อม ๆ กับตอบคำถามว่าระหว่างคู่ของห้องที่ถาม สามารถเดินไปหากันได้หรือไม่

คุณต้องใช้ API ในการอ่านคำถามและตอบคำถาม (ทั้งนี้มีเพื่อให้คุณตอบคำถามทันทีและป้องกันไม่ให้คุณใช้ข้อมูลของคำถามในอนาคตในการตอบคำถามปัจจุบัน)

การใช้ไลบรารี

คุณจะได้รับ include file “castle.h” (ซึ่งต้อง include อยู่ในโปรแกรมของคุณ) และ source file “castle.cpp” (ซึ่งต้องคอมไพล์ร่วมกับโปรแกรมของคุณ) ซึ่งมีฟังก์ชันดังต่อไปนี้

กลุ่มฟังก์ชันสำหรับอ่านข้อมูลเกี่ยวกับปราสาท

- castle_init(int& N, int& M, int& Q, int& Y) – สำหรับอ่านจำนวนห้อง N, จำนวนทางเดิน M, จำนวนเหตุการณ์ Q ที่คุณจะต้องจัดการ และ Y ระบุว่าคุณทราบข้อมูลเกี่ยวกับทางเดินธรรมดา-ทางเดินพิเศษหรือไม่ (1<=N<=100,000; N – 1 <= M<=N+9; 1<=Q<=100,000; 1<=Y<=2)
- castle_read_map(vector<int>& A, vector<int>& B) – จะคืนเวกเตอร์ที่มีสมาชิก M ตัวแทนข้อมูลทางเดิน โดยสำหรับ0<=i<=M-1 ทางเดินหมายเลข i ที่จะเชื่อมห้อง A[i] กับห้อง B[i] หมายเลขห้องและหมายเลขทางเดินจะเริ่มที่ 0

สำหรับทางเดินเหล่านี้ ถ้า Y = 1 ทางเดินที่ 0 ถึงทางเดินที่ N-2 จะเป็นทางเดินธรรมดา นอกจากนี้แล้วหมายเลขห้องในปราสาทยังเป็นตามเงื่อนไขนี้ ห้องที่เป็นรากคือห้องหมายเลข 0 จากนั้นห้องหมายเลข i ใด ๆ ที่ไม่ใช่ห้องในชั้นสุดท้าย จะมีทางเดินเชื่อมหมายเลข 2i กับ 2i+1 เชื่อมไปยังห้องหมายเลข 2(i+1) – 1 และ 2(i+1) และทางเดินตั้งแต่หมายเลข N-1 เป็นต้นไปจะเป็นทางเดินพิเศษ

ในกรณีที่ Y = 2 ทางเดินเชื่อมระหว่างห้องไม่จำเป็นต้องมีลักษณะตามที่กล่าวมาข้างต้น

คุณต้องเรียกฟังก์ชัน castle_init ตามด้วย castle_read_map ก่อนจะเริ่มทำงานอื่น ๆ ต่อไป

กลุ่มฟังก์ชันสำหรับอ่านเหตุการณ์และตอบคำถาม

- castle_read_event(int& T, int& X, int& Y) – อ่านเหตุการณ์ ถ้า T = 1 จะเป็นเหตุการณ์ที่ทางเดินพังลง โดย X จะเป็นหมายเลขทางเดิน (0 <= X <= M-1) และ Y=0 เสมอ ทางเดินใด ๆ จะไม่พังเกิน 1 ครั้ง (เมื่อพังลงจะพังไปตลอดกาลและไม่กลับมาใช้ได้อีก) ถ้า T = 2 เป็นคำถาม โดยถามว่าจากห้องหมายเลข X สามารถเดินไปถึงห้องหมายเลข Y ได้หรือไม่ (0<= X<=N-1; 0<=Y<=N-1)
- castle_answer(int R) – สำหรับเหตุการณ์ที่ T = 2 ให้คุณเรียก castle_answer เพื่อตอบ โดยให้ R = 1 ถ้าสามารถเดินได้ และให้ R = 0 ถ้าไม่สามารถเดินไปได้ ในกรณีนี้คุณต้องเรียก castle_answer ก่อนจะเรียก castle_read_event ครั้งต่อไป (ถ้า T=1 ต้องไม่เรียก castle_answer และสามารถ castle_read_event ต่อได้เลย)

คุณต้องเรียก castle_read_event จำนวน Q ครั้ง และคุณต้องจบการทำงานของโปรแกรมเอง

ตัวอย่างการทำงาน

สมมติให้ N = 3, M = 3, Q = 5 และ Y = 1 โดยมีทางเดินลัด 1 ทางเดิน เชื่อมห้องหมายเลข 1 กับ 2

call	output	คำอธิบาย
`castle_init(N,M,Q,Y)`	`N = 3, M = 3, Q = 5` และ `Y = 1`	
`castle_read_map(A,B)`	`A = [0,0,1]  B = [1,2,2]`	A[2], B[2] เป็นทางเดินลัด เชื่อมห้อง 1 กับ2
`castle_read_event(T,X,Y)`	`T = 2, X = 1, Y = 1`	ถามว่าห้อง 1 ไปถึงห้อง 1 ได้หรือไม่
`castle_answer(1)`		ตอบว่าเดินได้ (ถูกต้อง)
`castle_read_event(T,X,Y)`	`T = 1, X = 1, Y = 0`	ลบทางเดินหมายเลข 1 (ซึ่งเชื่อมห้อง 0 กับ 2)
`castle_read_event(T,X,Y)`	`T = 2, X = 2, Y = 0`	ถามว่าห้อง 2 ไปถึงห้อง 0 ได้หรือไม่
`castle_answer(1)`		ตอบว่าเดินได้ (ถูกต้อง)
`castle_read_event(T,X,Y)`	`T = 1, X = 2, Y = 0`	ลบทางเดินหมายเลข 2 (ซึ่งเชื่อมห้อง 1 กับ 2)
`castle_read_event(T,X,Y)`	`T = 2, X = 2, Y = 0`	
`castle_answer(0)`		ตอบว่าเดินไม่ได้ (ถูกต้อง)
`castle_read_event(T,X,Y)`	`T = 2, X = 0, Y = 1`	
`castle_answer(1)`		ตอบว่าเดินได้ (ถูกต้อง)

ปัญหาย่อย

- ปัญหาย่อย 1 (10%): N <= 1,000; Q <= 1,000
- ปัญหาย่อย 2 (10%): M = N – 1; Y = 1
- ปัญหาย่อย 3 (10%): M = N – 1; Y = 2
- ปัญหาย่อย 4 (10%): M = N; Y = 1
- ปัญหาย่อย 5 (10%): M = N; Y = 2
- ปัญหาย่อย 6 (25%): Y = 1
- ปัญหาย่อย 7 (25%): Y = 2

การใช้งานไลบรารีสำหรับทดสอบ

คุณสามารถดาวโหลดไลบรารีได้จาก (ดูในประกาศ) โค้ดในไลบรารีคุณสามารถแก้ไขได้เพื่อช่วยในการตรวจสอบโปรแกรม

ให้ป้อนข้อมูลเข้าในรูปแบบต่อไปนี้

N M Q Y

A[0] B[0]

A[1] B[1]

...

A[M-1] B[M-1]

T_0 X_0 Y_0

T_1 X_1 Y_1

...

T_{Q-1} X_{Q-1} Y_{Q-1}

ตัวอย่าง

```
3 3 6 1
0 1
0 2
1 2
2 1 1
1 1 0
2 2 0
1 2 0
2 2 0
2 0 1
```